

Sockets 2

Server Side Sockets

The client side can open a socket to “make a call” to a server – how does the server wait for incoming calls? The code below will “listen” for incoming connections on a particular port number – effectively waiting for an incoming connection on that port number.

On most operating systems, port numbers below 1024 are “privileged” and only the root/administrator user has permission to listen on them. This provides a primitive sort of security. If a regular user were allowed to listen on, say, port number 110 (which is used for email POP connections), then that user could pretend to be the POP server and fake people into giving over their passwords. Therefore important services tend to run on ports below 1024, and only special “privileged” users (e.g. root) are allowed to listen on those ports. This security scheme has proved to have disadvantages as well – the server software must run as a privileged user, and so if the server software is taken over, it's more damaging. (We'll look at these topics more in our security lectures.)

Setup

First, there are some basic setup operations that allocate the socket. The functions `socket()` and `sockaddr_in()` are as before. The `setsockopt()` function (optional) can be used to tell the server socket to re-use port numbers for separate connections – important if the server is going to run for many thousand client connections over time. The `bind()` function associates the server socket with the port number, but does not yet begin listening. The functions all return false on error.

Here's what a nicely decomposed `CreateServerSocket()` function might look like...

```
## Given a socket name to use and a port #,
## create a server socket so it's ready for the listen call
## Returns an error string on error, or "" on success.
## usage: $err = CreateServerSocket('SOCK', 80);
sub CreateServerSocket {
    my($sock, $port) = @_;

    socket($sock, PF_INET, SOCK_STREAM, 0) || return("socket $!");

    ## note: (a) this setsockopt() is not required, and
    ##        (b) that's an l not a 1 in the pack() call
    ##        i.e., pack(the lowercase L in quotes, the number 1)

    setsockopt($sock, SOL_SOCKET, SO_REUSEADDR, pack("l",1)) ||
        return("sockopt $!");

    bind($sock, sockaddr_in($port, INADDR_ANY)) || return("bind $!");
    autoflush $sock, 1;
    return("");
}
```

Listen

The `listen()` function gets the operating system to start listening for incoming connections on the bound port number. The first argument should be a server socket file handle that has been bound to the desired address. The call to `listen()` does not block. The OS will queue incoming socket connections up to some limit – maybe 5 or so. The constant, `SOMAXCONN` uses the system default for the number to queue. Returns false on error.

```
listen(SERVER, SOMAXCONN) || die "listen: $!";
```

Accept

The `accept()` function blocks, waiting for an incoming connection. The first argument should be an un-initiated file-handle name which will be set to the incoming client connection. The second argument should be the server socket from the `listen()` call. Returns an address struct identifying the incoming client, or false on error. It's probably a good idea to immediately set the new socket to unbuffered.

```
my $client_addr = accept(CLIENT, SERVER); ## blocks
autoflush CLIENT, 1;
```

Miscellaneous

The function `unpack_sockaddr_in()` returns an array with the numeric IP address and port number from the client address (example below). The function `inet_ntoa()` formats the IP address as a string. The `gethostbyaddr()` function does a reverse-dns lookup to get the dns name ("elaine33.stanford.edu") from the IP address (potentially slow). The function `getservbyname()` can be used to look up what port number goes with what service like this: `$port = getservbyname("finger", "tcp");`, although in our programs we will just hardwire the standard port number for the service we are implementing. On Unix, the mapping from services to port numbers is shown in the file `/etc/services`, and those are the mappings that `getservbyname()` reflects.

Server Example

This is a simple server example. This is a “single threaded” example – it processes the client connections one at a time. A more sophisticated server would fork off a separate process (or thread) to deal with each incoming client connection, so they could be handled in parallel (We will stop short of that level of socket complexity in CS193i.)

```
#!/usr/bin/perl
use Socket;
use FileHandle;

use strict 'vars';

my($server_port);
$server_port = 3456;

## Setup our SERVER socket
CreateServerSocket(SERVER, $server_port);
listen(SERVER, SOMAXCONN);
```

```

print "Server ready\n";

## Wait for incoming client connections
my $client_addr;
while ($client_addr = accept(CLIENT, SERVER)) {          ## accept() blocks

    autoflush CLIENT, 1;

    my ($port, $iaddr) = unpack_sockaddr_in($client_addr); ## extract ip and port
    my ($ip) = inet_ntoa($iaddr); ## ip as a string "171.64.64.250"
    my ($name) = gethostbyaddr($iaddr, AF_INET); ## lookup dns name (slow)

    print "Connection from $name [$ip] $port\n";

    ## print a line from the client (not dealing with EOLN issues)
    my($line);
    $line = <CLIENT>;    ## read a line from the client
    print $line;        ## print it locally
    print CLIENT "You said:$line"; ## send something back to the client
    close(CLIENT);
}

```

Client/Server Session

We use Telnet to play the client role – the typed-in parts are in bold. Note that at some point, the client packs what it thinks is its true IP Address into the payload and sends it across to the server. Bad software! Note that the server doesn't need the client to tell him what the client thinks is the true IP address. The server can just use `unpack_sockaddr_in(...)`, and `gethostbyaddr(...)`!

Server Transcript

```

saga10> ./servexample.pl
Server ready
Connection from solaria.Stanford.EDU [128.12.132.29] 54779
Hello there!
Connection from solaria.Stanford.EDU [128.12.132.29] 54782
Wow it actually worked!
Connection from solaria.Stanford.EDU [128.12.132.29] 54788
Hey my IP Address is 192.168.1.101
^C

```

Client Transcript

```

[traelysandr:~] ronyeh% telnet saga10 3456
Trying 171.64.15.140...
Connected to saga10.stanford.edu.
Escape character is '^]'.
Hello there!
You said:Hello there!
Connection closed by foreign host.
[traelysandr:~] ronyeh% telnet saga10 3456
Trying 171.64.15.140...
Connected to saga10.stanford.edu.
Escape character is '^]'.

```

```
Wow it actually worked!  
You said:Wow it actually worked!  
Connection closed by foreign host.  
[traelysandr:~] ronyeh% telnet saga10 3456  
Trying 171.64.15.140...  
Connected to saga10.stanford.edu.  
Escape character is '^]'.  
Hey my IP Address is 192.168.1.101  
You said:Hey my IP Address is 192.168.1.101  
Connection closed by foreign host.
```

You may find and play with the server example at
</usr/class/cs193i/WWW/hw/examples/servexample.pl>